

Time- and space-efficient evaluation of the complex exponential function using series expansion

Sergey V. Yakhontov
 Ph.D. in Computer Science
 Faculty of Mathematics and Mechanics
 Saint Petersburg State University
 Russian Federation
 SergeyV.Yakhontov@gmail.com
 Phone: +7-911-966-84-30
 14-Aug-2012

Abstract

An algorithm for the evaluation of the complex exponential function is proposed which is **quasi-linear in time and linear in space**. This algorithm is based on a modified binary splitting method for the hypergeometric series and a modified Karatsuba method for the fast evaluation of the exponential function. The time complexity of this algorithm is equal to that of the ordinary algorithm for the evaluation of the exponential function based on the series expansion: $O(M(n) \log(n)^2)$.

1. Introduction. In this paper we introduce a measure of the space complexity of calculations on a Schonhage machine [1] and give an upper bound for the time and space complexity of a proposed algorithm for the computation of the exponential function of a complex argument in each area $|z| \leq 2^p$ where p is a natural number, $p \geq 0$, on the Schonhage machine.

The Schonhage machine is in fact an ordinary computer. Therefore, we describe an algorithm which is quasi-linear in time and linear in space on an ordinary computer. Hence, the phrase ‘fast algorithm’ refers to evaluations on a Schonhage machine. In particular, the estimate $O(n \log(n) \log \log(n))$ refers to this machine [1].

Basic information on dyadic rational numbers and constructive real numbers and functions can be found in [2]. The notation **Sch(FQLINTIME//LINSPEACE)** will be used for the class of algorithms which are quasi-linear in time and linear in space on a Schonhage machine. Quasi-linear means that the complexity function is bounded by $O(n \log(n)^k)$ for some k .

From now on, n will denote the length of the record of accuracy 2^{-n} of dyadic rational approximations; x will be used for a real argument, z will be used for a complex argument. We will use $\log(k)$ for logarithms base 2.

The basic subject of interest is the algorithms for the evaluation of elementary functions based on series expansions, as such algorithms are important for practical computer science due to the relative simplicity of their implementation.

Algorithms for the fast evaluation of the exponential function and some other elementary functions with a time complexity of $O(M(n) \log(n)^2)$ are offered in [3] (where $M(n)$ denotes the complexity of multiplication of n -bit integers); the space used by

these algorithms is bounded by $O(n \log(n))$, as will be shown below. In [4], algorithms for the calculation of elementary functions, based on Taylor series, which are quasi-linear in time are considered; the space used by the algorithms from [4] is bounded by a quasi-linear function as the classical binary splitting method uses $O(n \log(n))$ space for the intermediate results.

That is, elementary functions are computable using quasi-linear time and quasi-linear space. There is a question: whether it is possible to evaluate elementary functions using quasi-linear in time and linear in space algorithms based on series expansions?

This paper shows that the answer to this question for the complex exponential function and some other complex elementary functions is Yes. We use the combination of two algorithms for the construction of an algorithm of complexity class **Sch(FQLINTIME//Linspace)** for the evaluation of the exponential function: a modified binary splitting method for the evaluation of the hypergeometric series, and a modified Karatsuba method [3] for the fast evaluation of the exponential function.

Note that the residual sum of the series (7) satisfies the following inequality:

$$|R_\nu(r)| < C2^{-(r \log(r)+m)};$$

here $r = m2^{-\nu+1}$. Therefore, $|R_\nu(r)| = O(2^{-m \log(m)})$ doesn't hold, and we cannot get the result about **FLinspace** computability of $\exp(x)$ from this equation.

2. Description of the computation model (machine Schonhage). This machine, introduced in [1], operates on symbols of the alphabet $\Sigma = \{0, 1, \dots, 2^{\lambda-1}\}$ and sequences of such symbols (we can take, for example, a constant λ equal to 32). The machine consists of arrays T_0, \dots, T_τ to read and write symbols from Σ , registers A, B, C, M for arithmetic operations, and a control unit CPU. The arrays are infinite in both directions. For each array there is a pointer p_i to the current symbol written in the array. The record $< p + j >$ means the symbol referenced by pointer $p + j$. There is also an additional register Y which is a pointer to the current operation, and an optional array S which serves as the stack of recursive calls; S is infinite in one direction. A bit register E acts as an overflow register for arithmetic operations.

A program for Schonhage consists of several modules written in the language TPAL, which is similar to an assembly language for a RISC processor. In TPAL there are commands for loading a symbol written in an array into a register, for reading a symbol from a register and writing it to an array, for increasing and decreasing the content of a register, the shift command, the call and return from a procedure commands, the jump to a label command, and the conditional jump command. Integers on which the machine operates are encoded as symbol sequences in the alphabet Σ :

$$a = a_0 + a_1 2^\lambda + a_2 (2^\lambda)^2 + \dots + a_{k-1} (2^\lambda)^{k-1}, \quad 0 \leq a_i \leq 2^\lambda - 1.$$

The sign bit is written in the symbol which is before the senior symbol a_{k-1} .

Schonhage can call procedures and perform recursive calls. After the return from a procedure the memory occupied by the parameters and local variables is released.

The time computational complexity of an algorithm on Schonhage is defined as the number of instructions in the language TPAL. Arithmetic operations on symbols and calls of procedures are counted as a constant number of steps. The memory used in an array during the calculation is defined as the maximum of the number of array elements involved in the calculation.

As constructive functions are functions that compute approximations of functions using approximations of arguments, we define the oracle machine Schonhage. This machine has some oracle functions that compute approximations of arguments; the machine calculates approximations of a function using these approximations of arguments. A request to an oracle is written in array T_0 as the record of an accuracy of

the computation; approximations of arguments are recorded in array T_0 too. A query to an oracle is treated as one operation in the time computational complexity of the oracle machine Schonhage.

Definition 1. *The space computational complexity of an algorithm on the oracle machine Schonhage is defined as the sum of the memory used for all the arrays plus the maximum of the memory used for the stack.*

3. Constructive complex numbers and functions. A complex number z' such that $|z - z'| \leq 2^{-n}$ is called an approximation of the complex number z with accuracy 2^{-n} .

Suppose that there are complex numbers $\omega = x + iy$, $\omega' = x' + iy'$ such that $|x - x'| \leq 2^{-(n+1)}$ and $|y - y'| \leq 2^{-(n+1)}$. Then

$$|\omega - \omega'| = \sqrt{(x - x')^2 + (y - y')^2} \leq \sqrt{2 \cdot 2^{-2(n+1)}} < 2^{-n}. \quad (1)$$

That is, to calculate an approximation of complex number ω with accuracy 2^{-n} it is sufficient to calculate an approximation of the real and imaginary parts of the complex number with accuracy $2^{-(n+1)}$.

We say that a sequence $\phi : \mathbf{N} \rightarrow \mathbf{D} \times \mathbf{D}$, where $\phi(n) = (\phi_x(n), \phi_y(n))$, \mathbf{D} is the set of dyadic rational numbers, converges dyadic-rationally to the complex number z if for any $n \in \mathbf{N}$ the following holds: $\text{prec}(\phi_x(n)) = n + 2$, $\text{prec}(\phi_y(n)) = n + 2$, and $|z(n) - z| \leq 2^{-n}$, where $z(n) = \phi_x(n) + i\phi_y(n)$. The set of all functions ϕ which converge dyadic-rationally to a number z is denoted by CF_z . A complex number z is called a CF constructive complex number if CF_z contains a computable function ϕ .

Definition 2. *A complex number $z \in \mathbb{C}$ is called a **Sch(FQLINTIME//Linspace)** constructive complex number if there exists a function $\phi \in CF_z$ which belongs to the class **Sch(FQLINTIME//Linspace)**.*

Let f be a function $f(z) : A \rightarrow \mathbb{C}$, where $A = \{z \in \mathbb{C} : |z| \leq R\}$ is an area in the set of complex numbers.

Definition 3. *The function $f(z)$ is called a **Sch(FQLINTIME//Linspace)** constructive complex function in the area A if for any z from this area there is a function ψ from $CF_{f(z)}$ which belongs to the class **Sch(FQLINTIME//Linspace)**.*

Note that to calculate the values of a constructive complex function of the argument $z = x + iy$ we need to specify functions $u(x, y) = \text{Re}(f(z))$ and $v(x, y) = \text{Im}(f(z))$, and in order to calculate these functions we need to have two oracle functions that correspond to the real and imaginary parts of the argument.

4. Binary splitting method. This method is used to calculate the values of series with rational coefficients, in particular, to calculate the hypergeometric series of the form

$$S = \sum_{i=0}^{\infty} \frac{a(i)}{b(i)} \prod_{j=0}^i \frac{p(j)}{q(j)},$$

where a , b , p , and q are polynomials with integer coefficients. Linearly convergent hypergeometric series are used to calculate many constants of analysis and elementary functions at rational points; this series is linearly convergent if its partial sum

$$S(\mu(k)) = \sum_{i=0}^{\mu(k)} \frac{a(i)}{b(i)} \prod_{j=0}^i \frac{p(j)}{q(j)}, \quad (2)$$

where $\mu(k)$ is a linear function of k , differs from the exact value by not more than 2^{-k} :

$$|S - S(\mu(k))| \leq 2^{-k}.$$

In its classical variant, the binary splitting method works as follows. Put $k_1 = \mu(k)$. We consider the partial sum (2) for some integers i_1 and i_2 , $0 \leq i_1 \leq k_1$, $0 \leq i_2 \leq k_1$, $i_1 \leq i_2$:

$$S(i_1, i_2) = \sum_{i=i_1}^{i_2} \frac{a(i)p(i_1) \dots p(i)}{b(i)q(i_1) \dots q(i)}.$$

We calculate $P(i_1, i_2) = p(i_1) \dots p(i_2)$, $Q(i_1, i_2) = q(i_1) \dots q(i_2)$, $B(i_1, i_2) = b(i_1) \dots b(i_2)$, and $T(i_1, i_2) = B(i_1, i_2)Q(i_1, i_2)S(i_1, i_2)$. If $i_1 = i_2$ then these values are calculated directly. Otherwise, the series is divided into two parts, left and right, and $P(i_1, i_2)$, $Q(i_1, i_2)$, and $B(i_1, i_2)$ are calculated for each part recursively. Then the values obtained are combined:

$$\begin{aligned} P(i_1, i_2) &= P_l P_r, & Q(i_1, i_2) &= Q_l Q_r, & B(i_1, i_2) &= B_l B_r, \\ T(i_1, i_2) &= B_r Q_r T_l + B_l P_l T_r. \end{aligned} \tag{3}$$

The algorithm starts with $i_1 = 0$, $i_2 = k_1$. After calculating $T(0, k_1)$, $B(0, k_1)$, and $Q(0, k_1)$, we divide $T(0, k_1)$ by $B(0, k_1)Q(0, k_1)$ to get the result with the given accuracy. We write out the binary splitting method explicitly.

Algorithm *BinSplit*.

Approximate value of the partial sum (2) with accuracy 2^{-k} .

Input: Record of accuracy 2^{-k} .

Output: Approximate value of (2) with accuracy 2^{-k} .

Description:

- 1) $k_1 := \mu(k)$;
- 2) $[P, Q, B, T] := \text{BinSplitRecurs}(0, k_1)$;
- 3) perform division $r := \frac{T}{BQ}$ with accuracy 2^{-k} ;
- 4) return result r .

This algorithm uses the following subalgorithm computing recursively the values P , Q , B , and T .

Algorithm *BinSplitRecurs*.

Calculation of P , Q , B , and T .

Input: Bounds i_1, i_2 of the interval.

Output: Tuple $[P(i_1, i_2), Q(i_1, i_2), B(i_1, i_2), T(i_1, i_2)]$.

Description:

- 1) if $i_1 = i_2$ then $T := a(i_1)p(i_1)$ and return tuple $[p(i_1), q(i_1), b(i_1), T]$;
- 2) $i_{mid} := \frac{i_1 + i_2}{2}$;
- 3) calculate $[P_l, Q_l, B_l, T_l] := \text{BinSplitRecurs}(i_1, i_{mid})$;
- 4) calculate $[P_r, Q_r, B_r, T_r] := \text{BinSplitRecurs}(i_{mid}, i_2)$;
- 5) $T := B_r Q_r T_l + B_l P_l T_r$ and return tuple $[P_l P_r, Q_l Q_r, B_l B_r, T]$.

The lengths of $T(0, k_1)$ and $B(0, k_1)Q(0, k_1)$ are proportional to $k \log(k)$; therefore the binary splitting method is quasi-linear in space; the time complexity of this algorithm is $O(M(k) \log(k)^2)$ [4].

5. Karatsuba's method for fast evaluation of $\exp(x)$. We consider the Taylor series of the real exponential function

$$\exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + \dots \quad (4)$$

at x_0 , $-\frac{1}{4} + 2^{-m} < x_0 < \frac{1}{4} - 2^{-m}$. We compute the value of this series with accuracy 2^{-n_3} , $n_3 > 7$, at the dyadic rational point x_m , $|x_m - x_0| \leq 2^{-m}$, $-\frac{1}{4} < x_m < \frac{1}{4}$. Let k be the smallest value such that

$$n_3 + 1 \leq 2^k, \quad m = 2^{k+1}. \quad (5)$$

We represent $x_m = \pm 0.00\alpha_3\alpha_4 \dots \alpha_m\alpha_{m+1}$ as

$$\begin{aligned} x_m &= \pm 0.00\alpha_3\alpha_4 + \pm 0.0000\alpha_5\alpha_6\alpha_7\alpha_8 + \dots + \pm 0.00 \dots 0a_{m-2^k+1}a_{m-2^k+2} \dots \alpha_m\alpha_{m+1} = \\ &= \frac{\beta_2}{2^4} + \frac{\beta_3}{2^8} + \frac{\beta_4}{2^{16}} + \dots + \frac{\beta_{k+1}}{2^m} = \gamma_2 + \gamma_3 + \dots + \gamma_{k+1}, \end{aligned}$$

where $\beta_2 = \pm\alpha_3\alpha_4$, $\beta_3 = \pm\alpha_5\alpha_6\alpha_7\alpha_8$, \dots , $\beta_{k+1} = \pm a_{m-2^k+1}a_{m-2^k+2} \dots \alpha_m\alpha_{m+1}$; $\gamma_\nu = \beta_\nu 2^{-2^\nu}$, $2 \leq \nu \leq k+1$; β_ν is a $2^{\nu-1}$ -digit number. We write $\exp(x_m)$ as a product:

$$\exp(x_m) = \exp(\gamma_2) \exp(\gamma_3) \dots \exp(\gamma_{k+1}). \quad (6)$$

In Karatsuba's method (FEE method, fast evaluation of the exponent; this method is also known as the Brent trick [5]) the values $\exp(\gamma_\nu)$ are then calculated using a Taylor series (4):

$$\exp(\gamma_\nu) = 1 + \frac{\beta_\nu}{1!2^{2^\nu}} + \frac{\beta_\nu^2}{2!2^{2 \cdot 2^\nu}} + \dots + \frac{\beta_\nu^r}{r!2^{r \cdot 2^\nu}} + R_\nu(r) = \xi_\nu + R_\nu(r); \quad (7)$$

here $r = m2^{-\nu+1}$. As for the residual sum, the following inequality is satisfied [3]:

$$|R_\nu(r)| < 2 \frac{|\beta_\nu|^{r+1}}{(r+1)! \cdot 2^{(r+1)2^\nu}},$$

then $|R_\nu(r)| < 2^{-m}$. The values ξ_ν are obtained from the formulas

$$\xi_\nu = \frac{a_\nu}{b_\nu}, \quad a_\nu = \xi_\nu b_\nu, \quad b_\nu = r!2^{r2^\nu},$$

where the integers a_ν are computed using a sequential process of grouping members of the series (7). Note that the formula for b_ν contains the factorial of r , and r varies from $m2^{-1}$ to $m2^{-k}$. Hence there are values which are proportional to $n_3!$, and therefore the length of the intermediate results is proportional to $n_3 \log(n_3)$.

6. Modification of the binary splitting method. We will modify the binary splitting method for the evaluation of the hypergeometric series (7) so that the calculations are in the class **Sch(FQLINTIME//Linspace)**.

We take $r = m2^{-\nu+2}$, $k_1 = \log(r)$, and $r_1 = \lceil \frac{r}{k_1} \rceil$ (k_1 is a natural number), and write the partial sum of (7) as

$$P(\gamma_\nu) = \sigma_1 + \tau_2[\sigma_2 + \tau_3[\sigma_3 + \dots + \tau_{k_1-1}[\sigma_{k_1-1} + \tau_{k_1}\sigma_{k_1}]]], \quad (8)$$

where

$$\begin{aligned}
\sigma_1 &= 1 + \frac{\beta_\nu}{1!2^{2^\nu}} + \frac{\beta_\nu^2}{2!2^{2 \cdot 2^\nu}} + \dots + \frac{\beta_\nu^{r_1-1}}{(r_1-1)!2^{(r_1-1) \cdot 2^\nu}}, \\
\tau_2 &= \frac{\beta_\nu^{r_1}}{r_1!2^{r_1 \cdot 2^\nu}}, \\
\sigma_2 &= 1 + \frac{\beta_\nu}{(r_1+1)2^{2^\nu}} + \frac{\beta_\nu^2}{(r_1+1)(r_1+2)2^{2 \cdot 2^\nu}} + \dots + \frac{\beta_\nu^{r_1-1}}{(r_1+1) \dots (2r_1-1)2^{(r_1-1) \cdot 2^\nu}}, \\
\tau_3 &= \frac{\beta_\nu^{r_1}}{(r_1+1) \dots 2r_1 2^{r_1 \cdot 2^\nu}}, \\
\sigma_3 &= 1 + \frac{\beta_\nu}{(2r_1+1)2^{2^\nu}} + \frac{\beta_\nu^2}{(2r_1+1)(2r_1+2)2^{2 \cdot 2^\nu}} + \dots + \frac{\beta_\nu^{r_1-1}}{(2r_1+1) \dots (3r_1-1)2^{(r_1-1) \cdot 2^\nu}} \\
&\dots
\end{aligned}$$

The σ_t are calculated by the classical binary splitting method for the sum (2), where

$$\begin{aligned}
\mu(k) &= r_1 - 1, \quad a(i) = 1, \quad b(i) = 1, \\
p(j) &= \begin{cases} 1 & j = 0 \\ \beta_\nu & j \neq 0 \end{cases}, \quad q(j) = \begin{cases} 1 & j = 0 \\ ((t-1)r_1 + j)2^{2^\nu} & j \neq 0 \end{cases}. \quad (9)
\end{aligned}$$

We estimate the computational complexity of the calculation of σ_t and τ_t in the following two lemmas.

Lemma 1. *The time complexity of the binary splitting method for the calculation of σ_t on the Schonhage machine is bounded above by $(r \log(r) + m) \log(m) \log \log(m)$; the space complexity is bounded above by $O(m)$, where m is given by (5).*

Proof. We consider an arbitrary maximal chain of recursive calls derived from the calculations in accordance with algorithm *BinSplitRekurs*. We consider pairs (P, i) , (Q, i) , (B, i) , and (T, i) where i is the number of an element of the chain of recursive calls and P , Q , B , and T are in the i th element of the chain. We suppose that the numbering in the chain begins with its deepest element: $i = 1, \dots, \varsigma$, where ς is the length of the chain, $\varsigma \leq \lceil \log(2r_1) \rceil$.

Using mathematical induction on i , we show that the length of the representation of T in the pair (T, i) satisfies $l(T) < 2^i l(u) + 2^i$, where $u = r2^{2^\nu}$. We note that $l(\beta_\nu) < l(u)$ since β_ν is a $2^{\nu-1}$ -digit integer. Next, $l(P) < 2^i l(u)$, $l(Q) < 2^i l(u)$ for pairs (P, i) , (Q, i) , since increasing i leads to doubling the length; B is always 1.

The induction begins with $i = 1$: $l(T) \leq l(u) < 2^1 l(u) + 2^1$. This inequality follows from (9) and the formula $T = a(i_1)p(i_1)$ for $(T, 1)$. Next, the induction step is

$$l(T) < 2^i l(u) + 2^i l(u) + 2^i + 1 \leq 2^{i+1} l(u) + 2^{i+1}.$$

This inequality follows from (3): there are two multiplications of numbers of length $2^i l(u)$ and $2^i l(u) + 2^i$ and one addition.

We note that, based on this inequality, T of the pair (T, ς) has length $O(m)$ because

$$l(T) < C_1 2^\varsigma l(u) \leq C_2 \frac{r}{\log(r)} (\log(r) + 2^\nu) \leq C_2 \left[r + \frac{m2^{-\nu+1}2^\nu}{\log(r)} \right] \leq C_3 m.$$

We estimate the time complexity of the calculation of σ_t . We must take into account the property of the complexity of integer multiplication: $2M(2^{-1}m) \leq M(m)$ (quasi-linear and polynomial functions satisfy this property). Since at a tree node of recursive calls at level i there are C_4 multiplications of $2^{\varsigma-i}$ numbers of length at most $2^i l(u) + 2^i$ and

$h \leq C \log(m)$, we obtain the following estimate for the number of operations required to calculate σ_t :

$$\begin{aligned} \text{Time}(\sigma_t) &\leq C_4 \sum_{i=1}^{\varsigma} 2^{\varsigma-i} M(2^i l(u) + 2^i) \leq C_5 \sum_{i=1}^{\varsigma} 2^{\varsigma-i} M(2^{i+1} l(u)) \\ &= C_5 \sum_{i=1}^{\varsigma} 2^{\varsigma-i} M(2^{\varsigma-(\varsigma-(i+1))} l(u)) \leq C_6 \sum_{i=1}^{\varsigma} 2^{\varsigma-i} 2^{-\varsigma+(i+1)} M(2^{\varsigma} l(u)) \\ &\leq C_7 \varsigma M(2^{\varsigma} l(u)) \leq C_8 \log(r) M\left(r + \frac{m}{\log(r)}\right) \end{aligned}$$

(here we use the inequality for $2^{\varsigma} l(u)$ from the estimate of $l(T)$ for the pair (T, ς)). The final division gives $O(M(m))$ operations. If we use the Schonhage–Strassen algorithm for integer multiplication, then

$$\begin{aligned} \text{Time}(\sigma_t) &\leq C_9 \left(r + \frac{m}{\log(r)}\right) \log(r) \log(m) \log \log(m) = \\ &= C_9 (r \log(r) + m) \log(m) \log \log(m). \end{aligned} \tag{10}$$

Now we estimate the space complexity of the computation of σ_t . At an element of a chain of recursive calls with number i , the amount of memory consumed for the temporary variables is $C_{10}(2^i l(u) + 2^i)$. Hence, we conclude that the amount of memory in all the simultaneously existing recursive calls in the chain is estimated as follows:

$$\text{Space}(\sigma_t) \leq \sum_{i=1}^{\varsigma} C_{10}(2^i l(u) + 2^i) \leq C_{11}(2^{\varsigma} l(u) + 2^{\varsigma}) \leq C_{12} m = O(m).$$

□

Lemma 2. *The time complexity of the calculation of τ_t using binary splitting method on a Schonhage machine is bounded above by $(r \log(r) + m) \log(m) \log \log(m)$; the space complexity is bounded above by $O(m)$ where m is given by (5).*

Proof. The estimates of the computational complexity of τ_t are the same as those for σ_t due to the fact that the inequalities in the proof of Lemma 1 are also suitable for τ_t (we can calculate the numerator and denominator of σ_t using the binary splitting method for products). □

We calculate approximate values $P(\gamma_\nu)^*$ with accuracy $2^{-(m+1)}$ by (8) using the following iterative process:

$$\begin{aligned} h_1(m_1) &= \sigma_{k_1}^*, \\ \hat{h}_i(m_1) &= \sigma_{k_1-i+1}^* + \tau_{k_1-i+2}^* h_{i-1}, \quad i = 1, \dots, k_1, \\ h_i(m_1) &= \hat{h}_i(m_1) + \varepsilon_i; \end{aligned} \tag{11}$$

for $i = k_1$ we set $P(\gamma_\nu)^* = h_{k_1}(m_1)$. Here $m_1 \geq m$ (m_1 will be chosen later), and σ_i^* and τ_i^* are approximations of σ_i and τ_i with accuracy 2^{-m_1} . The $h_i(m_1)$ are obtained by discarding bits $q_{m_1+1} q_{m_1+2} \dots q_{m_1+j}$ of numbers $\hat{h}_i(m_1)$ after the binary point starting with the $(m_1 + 1)$ th bit:

$$|\varepsilon_i| = |h_i(m_1) - \hat{h}_i(m_1)| = 0.0 \dots 0 q_{m_1+1} q_{m_1+2} \dots q_{m_1+j}, \tag{12}$$

and the sign of ε_i is the same as the sign of $\hat{h}_i(m_1)$ (it is clear that $|\varepsilon_i| < 2^{-m_1}$).

We note that the accuracy of $\exp(\gamma_\nu)$ is 2^{-m} since we compute $\xi_\nu^* = P(\gamma_\nu)^*$ with accuracy $2^{-(m+1)}$ and

$$|\exp(\gamma_\nu)^* - \exp(\gamma_\nu)| \leq |\xi_\nu^* - \xi_\nu| + |R_\nu(r)| < 2^{-(m+1)} + 2^{-(m+1)} = 2^{-m}.$$

Lemma 3. For every $i \in 1 \dots k_1$

$$|h_i(m_1)| < 2. \quad (13)$$

Proof. We apply mathematical induction on j for $h_j(m_1)$ using the estimates

$$|\sigma_i| < \exp(\gamma_\nu) < \frac{4}{3}, \quad \tau_i < \gamma_\nu^{r_1} \leq \frac{1}{4}.$$

The induction base for j is $j = 1$: $|h_1(m_1)| \leq \sigma_{k_1} + 2^{-m_1} < 2$. The induction step for $(j + 1) \geq 2$:

$$\begin{aligned} |h_{j+1}(m_1)| &= |\sigma_{k_1-(j+1)+1}^* + \tau_{k_1-(j+1)+2}^* h_j + \varepsilon_{j+1}| \\ &\leq \frac{4}{3} + \left[\frac{1}{4} + 2^{-m_1} \right] 2 + 2^{-m_1} < 2. \end{aligned}$$

□

Lemma 4. The error of the calculation of $h_{k_1}(m_1)$ using the scheme (11) is estimated to be

$$\Delta(k_1, m_1) < 2^{-m_1+k_1}.$$

Proof. We put

$$H_1 = \sigma_{k_1}, \quad H_i = \sigma_{k_1-i+1} + \tau_{k_1-i+2} H_{i-1}, \quad \eta(i, m_1) = |h_i(m_1) - H_i|.$$

We use mathematical induction for $\eta(j, m_1)$ on j . The induction base for j is 1:

$$\eta(1, m_1) = |h_1(m_1) - H_1| = |\sigma_{k_1}^* - \sigma_{k_1}| < 2^{-m_1+1}.$$

The induction step is $(j + 1) \geq 2$:

$$\begin{aligned} \eta(j+1, m_1) &= |\sigma_{k_1-(j+1)+1}^* + \tau_{k_1-(j+1)+2}^* h_j(m_1) + \varepsilon_{j+1} - \sigma_{k_1-(j+1)+1} - \tau_{k_1-(j+1)+2} H_j| \\ &< |\tau_v^* h_j(m_1) - \tau_v h_j(m_1) + \tau_v h_j(m_1) - \tau_v H_j| + 2 \cdot 2^{-m_1} \\ &\leq 2^{-m_1} h_j(m_1) + 2^{-2} \eta(j, m_1) + 2 \cdot 2^{-m_1}. \end{aligned}$$

Since (13), $|h_j(m_1)| < 2$. By the induction hypothesis, $\eta(j, m_1) < 2^{-m_1+j}$, and so we get

$$\eta(j+1, m_1) < 2 \cdot 2^{-m_1} + 2^{-2} 2^{-m_1+j} + 2 \cdot 2^{-m_1} < 2^{-m_1+(j+1)}.$$

From $\Delta(k_1, m_1) = \eta(k_1, m_1)$ we now obtain the required inequality. □

Lemma 4 implies that it is sufficient to take $m_1 = 2m + 1$ to compute $P(\gamma_\nu)$ with an accuracy of $2^{-(m+1)}$.

We denote the algorithm for the calculation of the hypergeometric series using scheme (11) by *RLinSpaceBinSplit* (linear space binary splitting).

Algorithm *RLinSpaceBinSplit*.

The approximate value of the hypergeometric series.

Input: Record of the accuracy 2^{-m} .

Output: The approximate value of (7) with accuracy 2^{-m} .

Description:

- 1) $m_1 := 2m + 1$;

- 2) $h := \sigma_{k_1}^*$ (using the classical binary splitting method with accuracy 2^{-m_1});
- 3) make a loop through i from 2 to k_1 :
 - a) calculate $v_1 := \sigma_{k_1-i+1}^*$ with accuracy 2^{-m_1} using the classical binary splitting method and $v_2 := \tau_{k_1-i+2}^*$ with accuracy 2^{-m_1} ,
 - b) calculate $\widehat{h} := v_1 + v_2 h$,
 - c) assign value \widehat{h} to h rounded in accordance with (12);
- 4) write h on exit.

We estimate the time computational complexity of this algorithm on a Schonhage machine, taking into account that m depends linearly on n_3 :

- $\log(r)$ computations of σ_t give the following (from inequality (10)):

$$\begin{aligned} \text{Time}(\text{all}(\sigma_t)) &\leq \sum_{\nu=2}^{\log(m)} C_9(r \log(r) + m) \log(m) \log \log(m) \leq \\ &\leq C_{13}(m \log(m)^2 \log \log(m)) = O(M(n_3) \log(n_3)); \end{aligned}$$

- $O(\log(n_3))$ computations of τ_t give $O(M(n_3) \log(n_3))$;
- $O(\log(n_3))$ multiplications of numbers of the length $O(n_3)$ give $O(M(n_3) \log(n_3))$;

in total we obtain $O(M(n_3) \log(n_3))$. The space complexity of the modified binary splitting method *RLinSpaceBinSplit* is $O(n_3)$ since in all calculations in this algorithm we process numbers of length $O(n_3)$.

Proposition 1. *The modified binary splitting algorithm *RLinSpaceBinSplit* belongs to the class **Sch(FQLINTIME//Linspace)**.*

7. Modification of FEE. We construct a modification of the method FEE so that our calculations are in class *Sch(FQLINTIME//Linspace)*.

In the classical algorithm FEE the values $\exp(x_m)^*$ are calculated by (6) using a pairwise summation of the numbers $\exp(\gamma_i)^*$; in FEE we need to keep in memory $O(\log(\log(n_3)))$ numbers of length $O(n_3)$ and, as already mentioned, $n_3!$ values are processed in this algorithm, so it does not have linear space complexity.

We calculate $\exp(x_m)^*$ with accuracy 2^{-n_3} using the following iterative process:

$$\begin{aligned} h_2(m) &= \exp(\gamma_2)^*, \\ \widehat{h}_i(m) &= h_{i-1}(m) \exp(\gamma_i)^*, \quad i = 2, \dots, k+1, \\ h_i(m) &= \widehat{h}_i(m) + \varepsilon_i; \end{aligned} \tag{14}$$

for $i = k+1$ we put $\exp(x_m)^* = h_{k+1}(m)$. Here $\exp(\gamma_i)^*$ are approximations for the values $\exp(\gamma_i)$ with accuracy 2^{-m} obtained from formula (7). The values $h_i(m)$ are obtained by discarding bits $q_{m+1}q_{m+2} \dots q_{m+t}$ of the numbers $\widehat{h}_i(m)$ after the binary point starting with the $(m+1)$ th bit, i.e.,

$$|\varepsilon_i| = |h_i(m) - \widehat{h}_i(m)| = 0.0 \dots 0 q_{m+1} q_{m+2} \dots q_{m+t}, \tag{15}$$

and the sign of ε_i is the same as the sign of $\widehat{h}_i(m)$ (it is clear that $|\varepsilon_i| < 2^{-m}$).

Lemma 5. *For every $i \in 2 \dots k+1$*

$$|h_i(m)| < 2^{i-1}. \tag{16}$$

Proof. We apply mathematical induction on j for $h_j(m)$. The induction base is $j = 2$:

$$|h_2(m)| \leq |\exp(\gamma_2)| + 2^{-m} < \frac{3}{2} + 2^{-16} < 2^{2-1}$$

(here we take into account that $|\gamma_i| < \frac{1}{4}$, $m \geq 16$). The induction step is $(j + 1) \geq 3$:

$$\begin{aligned} |h_{j+1}(m)| &= |h_j(m) \exp(\gamma_{j+1})^* + \varepsilon_{j+1}| < h_j(m)(\exp(\gamma_{j+1}) + 2^{-m}) + 2^{-m} \\ &< 2^{j-1} \left[\frac{3}{2} + 2^{-m} \right] + 2^{-m} < 2^j. \end{aligned}$$

□

Lemma 6. *The error of the calculation of $h_{k+1}(m)$ using scheme (14) is estimated to be*

$$\Delta(k + 1, m) < 2^{-m+2(k+1)}.$$

Proof. We put

$$H_2 = \exp(\gamma_2), \quad H_i = \exp(\gamma_2) \exp(\gamma_3) \dots \exp(\gamma_i), \quad \eta(i, m) = |h_i(m) - H_i|.$$

We use mathematical induction for $\eta(j, m)$ on j . The induction base is $j = 2$:

$$\eta(2, m) = |h_2(m) - H_2| = |\exp(\gamma_2)^* - \exp(\gamma_2)| \leq 2^{-m} < 2^{-m+2(1+1)}.$$

The induction step is $(j + 1) \geq 3$:

$$\begin{aligned} \eta(j + 1, m) &= |h_j(m) \exp(\gamma_{j+1})^* + \varepsilon_{j+1} - H_j \exp(\gamma_{j+1})| \\ &< |h_j(m) \exp(\gamma_{j+1})^* - h_j(m) \exp(\gamma_{j+1}) + \\ &\quad h_j(m) \exp(\gamma_{j+1}) - H_j \exp(\gamma_{j+1})| + 2^{-m} \\ &\leq h_j(m) 2^{-m} + \eta(j, m) \exp(\gamma_{j+1}) + 2^{-m}. \end{aligned}$$

Since (16), $|h_j(m)| < 2^{j-1}$. Then by the induction hypothesis, $\eta(j, m) < 2^{-m+2j}$, and so we get

$$\eta(j + 1, m) < 2^{j-1} 2^{-m} + [2^{-m+2j}] \frac{3}{2} < 2^{-m+2(j+1)}.$$

From $\Delta(k + 1, m) = \eta(k + 1, m)$, we now get the required inequality. □

Lemma 6 implies that the accuracy 2^{-m} of the calculation of $\exp(\gamma_i)^*$ is sufficient to calculate $\exp(x_m)^*$ with accuracy 2^{-n_3} , since

$$\begin{aligned} -m + 2(k + 1) &\leq -n_3 \Rightarrow 2^{k+1} - 2(k + 1) \geq n_3 \quad \text{and} \\ 2^{k+1} - 2(k + 1) &> 2^{k+1} 2^{-1} = 2^k \geq n_3 + 1 \end{aligned}$$

(here we recall that $n_3 + 1 \leq 2^k$).

We denote the algorithm of the calculation of the real exponential function using scheme (14) by *RLinSpaceFEE* (linear space fast exponential evaluation).

Algorithm *RLinSpaceFEE*.

The approximate value of the real exponential function on the interval $[-\frac{1}{8}, \frac{1}{8}]$.

Input: Record of the accuracy 2^{-n_3} .

Output: The approximate value $\exp(x)$ with accuracy 2^{-n_3} .

Oracles: ϕ_x .

Description:

- 1) $h := \exp(\gamma_2)^*$ (using algorithm *RLinSpaceBinSplit* with accuracy 2^{-m});
- 2) make a loop through i from 3 to $k + 1$:
 - a) calculate $v_1 := \exp(\gamma_i)^*$ using algorithm *RLinSpaceBinSplit* with accuracy 2^{-m} ,
 - b) calculate $\hat{h} := h \cdot v_1$,
 - c) assign the value \hat{h} to h rounded in accordance with (15);
- 3) write h on exit.

The time complexity of this algorithm on a Schonhage is $O(M(n_3) \log(n_3)^2)$ as the algorithm of the calculation of the hypergeometric series *RLinSpaceBinSplit* uses $O(M(n_3) \log(n_3))$ operations, and in scheme (14) there are $O(\log(n_3))$ such calculations and $O(\log(n_3))$ multiplications of numbers of the length $O(n_3)$; the space complexity of *RLinSpaceFEE* is $O(n_3)$ since in all the calculations in this algorithm numbers of length $O(n_3)$ are used.

Proposition 2. *The modified FEE algorithm *RLinSpaceFEE* for the calculation of the exponential function belongs to the class **Sch(FQLINTIME//Linspace)**.*

9. Calculation of the real function $\exp(x)$. Let p be a positive integer, $p \geq 0$. We compute the function $\exp(x)$ with accuracy 2^{-n} in the interval $[-2^p, 2^p]$.

We perform the multiplicative reduction of the interval of the complex argument. Namely, we take an integer $s = 2^{p+3}$ and $x' = \frac{x}{s}$; then $|x'| = \frac{|x|}{s}$, i.e., x' is in the interval $[-2^{-3} \leq x' \leq 2^{-3}]$. Thus the calculation of $\exp(x)$ is reduced to the computation of $\exp(x')$ and then we raise this value to the power s to get the result. It is easy to see that the dependency function for n_1 of the accuracy of $\exp(x')$ is $n_1 = L(n) + C(p)$, where $L(n)$ is a linear function of n , and $C(p)$ is a constant which is independent of p (constant in the sense that it doesn't depend on n).

Put $m \geq n_1 + 3$. We then have: $x_m = x_0 + \theta_1 2^{-m}$, $|\theta_1| \leq 1$; $|x_0| \leq 2^{-3}$, and

$$|\exp(x_0) - \exp(x_m)| = |\exp(x_0) - \exp(x_0) \exp(\theta_1 2^{-m})| = \exp(x_0) |1 - \exp(\theta_1 2^{-m})|.$$

Since $\exp(\theta_1 2^{-m}) < \frac{1}{1-2^{-m}}$, $|\exp(\theta_1 2^{-m}) - 1| < \frac{2^{-m}}{1-2^{-m}} < 2^{-m+1}$. Therefore we have the estimate

$$|\exp(x_0) - \exp(x_m)| < 2 \cdot 2^{-m+1} = 2^{-m+2} \leq 2^{-(n_1+1)},$$

which shows provided the accuracy of the calculation of x_m is better than $2^{-(n_1+3)}$, then one achieves an accuracy 2^{-n_2} , $n_2 = n_1 + 1$ for $\exp(x_0)$. Since $m = 2^{k+1}$, this condition is satisfied.

Now we need to keep in mind that we calculate the approximate value of $\exp(x_m)^*$. If this approximation is calculated with accuracy 2^{-n_3} , $n_3 = n_1 + 1$, then

$$\begin{aligned} |\exp(x_0) - \exp(x_m)^*| &\leq |\exp(x_0) - \exp(x_m)| + |\exp(x_m) - \exp(x_m)^*| \\ &< 2^{-(n_1+1)} + 2^{-(n_1+1)} = 2^{-n_1}. \end{aligned}$$

This implies that we can take $n_3 = n_1 + 1$ in algorithm *RLinSpaceFEE*.

We are now ready to describe the algorithm.

Algorithm *RLinSpaceExpValue*.

The approximate value of the complex exponential function.

Input: Record of the accuracy 2^{-n} .

Output: The approximate value $\exp(z)$ with accuracy 2^{-n} .

Parameters: ϕ_x for the argument x .

Oracles: Constant p .

Description:

- 1) $n_1 := L(n) + C(p)$;
- 2) $n_3 := n_1 + 1$;
- 3) calculate k, m so that (5) holds;
- 4) $p_1 := p + 3$
- 5) $s := 2^{p_1}$;
- 6) compute $x^* := \phi_x(\max(1, m - p_1))$;
- 7) perform the reduction of the interval: $(x^*)' = \frac{x^*}{s}$ (the accuracy of the arguments will be 2^{-m});
- 8) using algorithm *RLinSpaceFEE*, calculate $v := \exp(x^*)^*$ with accuracy 2^{-n_3} ;
- 9) write complex number v^s to the output.

The properties of algorithms *TLinSpaceBinSplit* and *TLinSpaceFEE* allow us assert the following propositions.

Proposition 3. *Algorithm* *RLinSpaceExpValue* *of the calculation of the complex exponential function belongs to the class* **Sch(FQLINTIME//Linspace)**.

The estimates of the computational complexity of algorithm *RLinSpaceExpValue* on the Schonhage machine are the same as those for algorithm *RLinSpaceFEE*: that is, the time complexity is $O(M(n_3) \log(n_3)^2)$ and the space complexity is $O(n_3)$. If we use the Schonhage–Strassen algorithm for integer multiplication, then the time complexity of algorithm *RLinSpaceExpValue* is bounded above by $O(n_3 \log(n_3)^3 \log \log(n_3))$.

Proposition 4. *The real function* $\exp(x)$ *is a* **Sch(FQLINTIME//Linspace)** *constructive real function in any interval* $[-2^p, 2^p]$.

8. Calculation of the function $\exp(\mathbf{i} \cdot y)$. It is easy to show that all the algorithms, lemmas, and estimates can be formulated for the evaluation of the function $\exp(\mathbf{i} \cdot y)$.

1. Calculate (2), where $p(j) = \mathbf{i} \cdot p_{\text{real}}(j)$; P, Q, B are integers, and T is complex. The estimates of the computational complexity are the same as the estimates in Lemmas 1 and 2. The $h_i(m)$ are obtained by discarding bits $q_{m_1+2}q_{m_1+3} \dots q_{m_1+j}$ of the numbers $\hat{h}_i(m_1)$ after the binary point starting with the $(m_1 + 2)$ th bit. Lemmas 3 and ?? are true for the new scheme. Algorithm *CLinSpaceBinSplit* is the same as *RLinSpaceFEE*.
2. In the FEE method, we calculate

$$\exp(\mathbf{i} \cdot x_m) = \exp(\mathbf{i} \cdot \gamma_2) \exp(\mathbf{i} \cdot \gamma_3) \dots \exp(\mathbf{i} \cdot \gamma_{k+1}).$$

The estimate for $|R_\nu(r)|$ is the same as that for $\exp(x)$, that is, the series $Eq : \text{RealExp} : \text{GammaNuSeries}$ converges linearly for $\exp(\mathbf{i} \cdot y)$.

3. In the formulas for σ_i and τ_i , we use $\mathbf{i} \cdot \beta_\nu$ and we use algorithm *CLinSpaceBinSplit* for the computation of $P(\mathbf{i} \cdot \gamma_\nu)^*$.

4. In the scheme for the computation of $P(\mathbf{i} \cdot \gamma_\nu)^*$, the $h_i(m)$ are obtained by discarding bits $q_{m_1+2}q_{m_1+3} \dots q_{m_1+j}$ of the numbers $\widehat{h}_i(m_1)$ after the binary point starting with the $(m_1 + 2)$ th bit; Lemmas 5 and 5 are true for the new scheme; algorithm *CLinSpaceFEE* is the same as *RLinSpaceFEE*.

Denote the algorithm for the computation of $\exp(\mathbf{i} \cdot y)$ by *CLinSpaceExpValue*.

9. Calculation of the complex function $\exp(z)$. Let p be a positive integer, $p \geq 0$. We compute the function $\exp(z)$ with accuracy 2^{-n} in the area $|z| \leq 2^p$ (we have $|x| \leq 2^p$, $|y| \leq 2^p$).

We perform the multiplicative reduction of the interval of the complex argument. Namely, we take an integer $s = 2^{p+3}$ and $z' = \frac{z}{s}$; then $|z'| = \frac{|z|}{s}$, i.e., z' is in the area $|z'| \leq 2^{-3}$ and both $|x| \leq 2^{-3}$ and $|y| \leq 2^{-3}$. Thus the calculation of $\exp(z)$ is reduced to the computation of $\exp(z')$ and then we raise this value to the power s to get the result. It is easy to see that the dependency function for n_1 of the accuracy of $\exp(z')$ is $n_1 = L(n) + C(p)$, where $L(n)$ is a linear function of n , and $C(p)$ is a constant which is independent of p (constant in the sense that it doesn't depend on n).

Next we consider the function $\exp(z)$ in the area $|z| \leq 2^{-3}$. Put $\zeta_x = \exp(x)$ and $\zeta_y = \exp(\mathbf{i} \cdot y)$. According to (1), we need to calculate ζ_x and ζ_y with accuracies of 2^{-n_2} , $n_2 = n_1 + 1$ respectively, in order to calculate $\exp(z)$ with an accuracy of 2^{-n_1+1} .

Put $m \geq n_1 + 3$. We have the following: $x_m = x_0 + \theta_1 2^{-m}$, $|\theta_1| \leq 1$; at that $|x_0| \leq 2^{-3}$ and

$$|\exp(x_0) - \exp(x_m)| = |\exp(x_0) - \exp(x_0) \exp(\theta_1 2^{-m})| = \exp(x_0) |1 - \exp(\theta_1 2^{-m})|.$$

Since $\exp(\theta_1 2^{-m}) < \frac{1}{1-2^{-m}}$, $|\exp(\theta_1 2^{-m}) - 1| < \frac{2^{-m}}{1-2^{-m}} < 2^{-m+1}$, and therefore we have the estimate

$$|\exp(x_0) - \exp(x_m)| < 2 \cdot 2^{-m+1} = 2^{-m+2} \leq 2^{-(n_1+1)},$$

which shows that if accuracy of the calculation of x_m is better than $2^{-(n_1+3)}$, then one achieves an accuracy of $2^{-(n_1+1)}$ for $\exp(x_0)$. Since $m = 2^{k+1}$, this condition is satisfied. A similar estimate can be obtained for $|\exp(\mathbf{i} \cdot y_0) - \exp(\mathbf{i} \cdot y_m)|$.

Now we need to keep in mind that we calculate the approximate value of $\exp(z_m)^*$. If this approximation is calculated with accuracy 2^{-n_3} , $n_3 = n_1 + 1$, then

$$\begin{aligned} |\exp(z_0) - \exp(z_m)^*| &\leq |\exp(z_0) - \exp(z_m)| + |\exp(z_m) - \exp(z_m)^*| \\ &< 2^{-(n_1+1)} + 2^{-(n_1+1)} = 2^{-n_1}. \end{aligned}$$

This implies that we can take $n_3 = n_1 + 1$ in algorithms *RLinSpaceFEE* and *CLinSpaceFEE*.

We are now ready to describe the basic algorithm.

Algorithm *LinSpaceExpValue*.

The approximate value of the complex exponential function.

Input: Record of the accuracy 2^{-n} .

Output: The approximate value $\exp(z)$ with accuracy 2^{-n} .

Parameters: ϕ_x and ϕ_y for the argument $z = x + iy$.

Oracles: Constant p .

Description:

- 1) $n_1 := L(n) + C(p)$;
- 2) $n_3 := n_1 + 1$;

- 3) calculate k, m so that (5) holds;
- 4) $p_1 := p + 3$
- 5) $s := 2^{p_1}$;
- 6) compute $x^* := \phi_x(\max(1, m - p_1))$, $y^* := \phi_y(\max(1, m - p_1))$;
- 7) perform the reduction of the interval: $(x^*)' = \frac{x^*}{s}$, $(y^*)' = \frac{y^*}{s}$ (the accuracy of the arguments will be 2^{-m});
- 8) using algorithm *RLinSpaceFEE*, calculate $v_1 := \zeta_x^*$ with accuracy 2^{-n_3} ; using algorithm *CLinSpaceFEE*, calculate $v_2 := \zeta_y^*$; here the arguments are $(x^*)'$, $(y^*)'$;
- 9) write to the output the complex number $(v_1 + i \cdot v_2)^s$.

The properties of algorithms *RLinSpaceBinSplit*, *RLinSpaceFEE*, *CLinSpaceBinSplit*, and *CLinSpaceFEE* allow us to assert the following propositions.

Proposition 5. *Algorithm *CLinSpaceExpValue* of the calculation of the complex exponential function belongs to the class **Sch(FQLINTIME//Linspace)**.*

The estimates of the computational complexity of algorithm *CLinSpaceExpValue* on a Schonhage machine are the same as those of algorithms *RLinSpaceFEE* and *CLinSpaceFEE*; that is, the time complexity is $O(M(n_3) \log(n_3)^2)$ and the space complexity is $O(n_3)$. If we use the Schonhage–Strassen algorithm for integer multiplication, then the time complexity of algorithm *CLinSpaceExpValue* is bounded above by $O(n_3 \log(n_3)^3 \log \log(n_3))$.

Theorem 1. *The complex function $\exp(z)$ is a **Sch(FQLINTIME//Linspace)** constructive complex function in any area $|z| \leq 2^p$.*

10. Computation of the complex functions $\sin(z)$, $\cos(z)$, $\text{sh}(z)$, $\text{ch}(z)$.
Based on the formulas for the trigonometric functions

$$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i} = i \frac{e^{iz} - e^{-iz}}{-2}, \quad \cos(z) = \frac{e^{iz} + e^{-iz}}{2},$$

we obtain the following:

Proposition 6. *The complex function $\sin(z)$ is a **Sch(FQLINTIME//Linspace)** constructive complex function in any area $|z| \leq 2^p$.*

Proposition 7. *The complex function $\cos(z)$ is a **Sch(FQLINTIME//Linspace)** constructive complex function in any area $|z| \leq 2^p$.*

The following two propositions also follow directly from the formulas for the hyperbolic sine and cosine:

$$\text{sh}(z) = \frac{e^z - e^{-z}}{2}, \quad \text{ch}(x) = \frac{e^z + e^{-z}}{2}.$$

Proposition 8. *The complex function $\text{sh}(z)$ is a **Sch(FQLINTIME//Linspace)** constructive complex function in any area $|z| \leq 2^p$.*

Proposition 9. *The complex function $\text{ch}(z)$ is a **Sch(FQLINTIME//Linspace)** constructive complex function in any area $|z| \leq 2^p$.*

11. Conclusion. Constructed algorithm *CLinSpaceExpValue* can be used in computer science as the basis of the **Sch(FQLINTIME//LINSPLACE)** constructive complex functions $\exp(z)$, $\sin(z)$, $\cos(z)$, $\operatorname{sh}(z)$, $\operatorname{ch}(z)$, defined on the set of **Sch(FQLINTIME//LINSPLACE)** constructive complex numbers.

Note also that if we use a simple recursive method for integer multiplication with time complexity $O(n^{\log(3)})$, then the time complexity of algorithm *CLinSpaceExpValue* is $O(n^{\log(3)} \log(n)^2)$.

As future research plans, we could note the problem of the construction of algorithms based on series expansions for the **Sch(FQLINTIME//LINSPLACE)** computable analogues of other elementary functions as well as the importance of the probably more difficult problem of the construction of computable analogues of elementary functions (also based on algorithms that use series expansions) with a time complexity of $O(n \log(n)^k)$, $k \leq 3$, and with linear space complexity.

References

- [1] Schonhage A., Grotefeld A. F. W, Vetter E. *Fast Algorithms. A Multitape Turing Machine Implementation.* // Germany: Brockhaus, 1994.
- [2] Ko K. *Complexity Theory of Real Functions.* // Boston: Birkhauser, 1991.
- [3] Karatsuba E. A. "Fast evaluation of transcendental functions." // *Problems of Information Transmission.* Vol. 27, Issue 4, 1991. pp. 76–99. (in Russian).
- [4] Haible B., Papanikolaou T. Fast multiple-precision evaluation of series of rational numbers. // *Proc. of the Third Intern. Symposium on Algorithmic Number Theory. June 21–25, 1998.* pp. 338–350.